GitHub depolarını Çatallama ve Klonlama

Çatal ve klon arasındaki farkın ne olduğunu merak ediyorsanız endişelenmeyin - GitHub'a başlarken bu doğal bir sorudur. Bu okumamızda bu soruya cevap vermenin yanı sıra hangisini ne zaman kullanmanız gerektiğini ve çatal ve klonlama işlemlerini nasıl yapacağınızı netleştireceğiz. Yerel, uzak, kaynak ve yukarı akış gibi depo (repo) terminolojisinin gizemini çözmenin yanı sıra. Ayrıca, alma, itme, çekme ve çekme isteklerini kullanarak çatallama, klonlama, değişiklik yapma ve çeşitli depoları senkronize tutmayı içeren tipik bir iş akışını inceleyeceğiz.

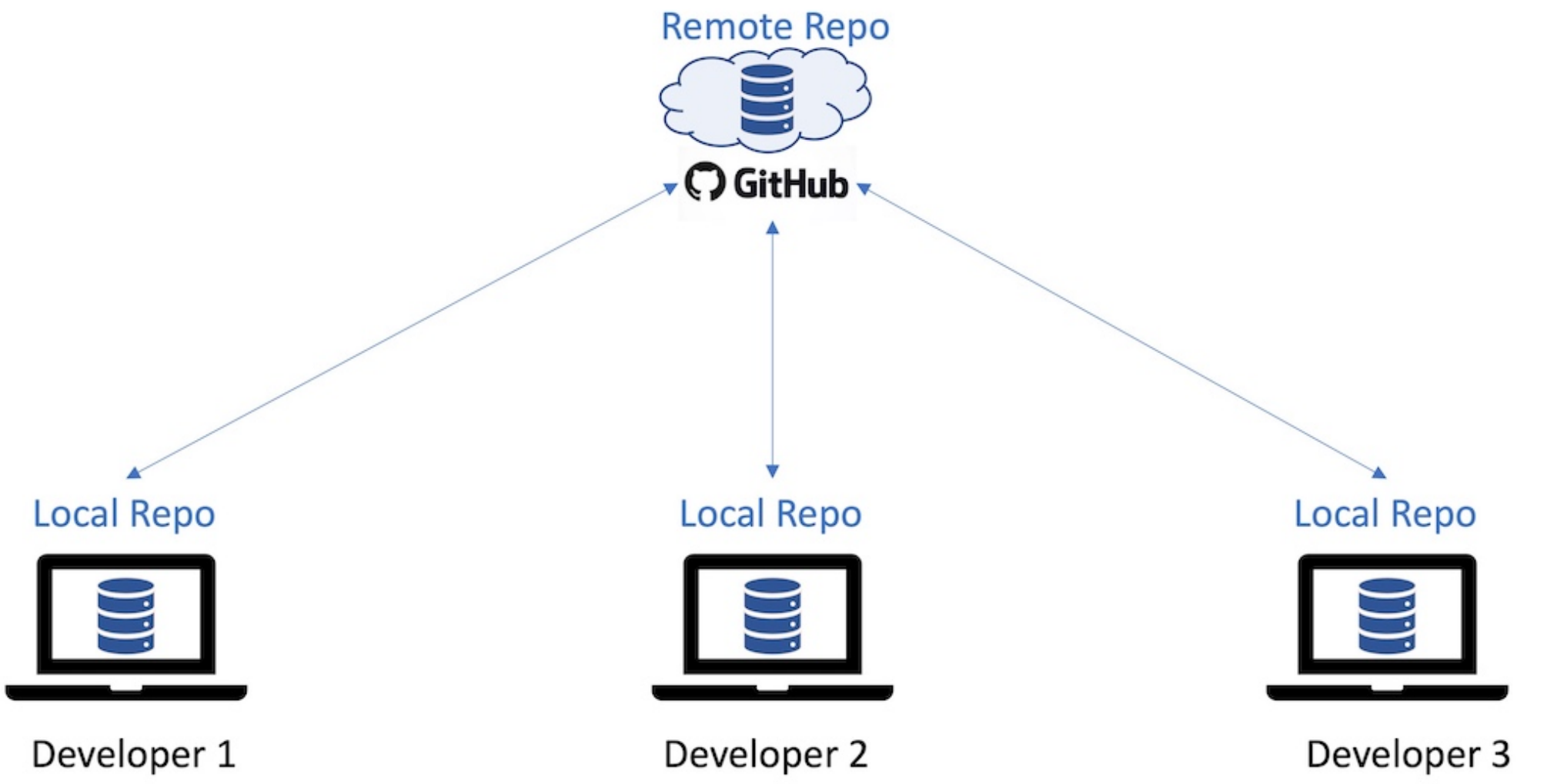
hedefler

Bu okumayı tamamladıktan sonra şunları yapabileceksiniz:

- Depoları çatallama ve klonlama arasında ayrım yapın
- Depoların nasıl çatallanabileceğini ve klonlanabileceğini açıklayın
- Ne zaman çatallanmanız ve klonlamanız gerektiğini belirleyin
- Yerel, uzak, kaynak ve yukarı akış havuzları gibi terminolojiyi tanımlayın
- Push, fetch ve pull kullanarak dağıtılmış depoları nasıl senkronize tutacağınızı açıklayın

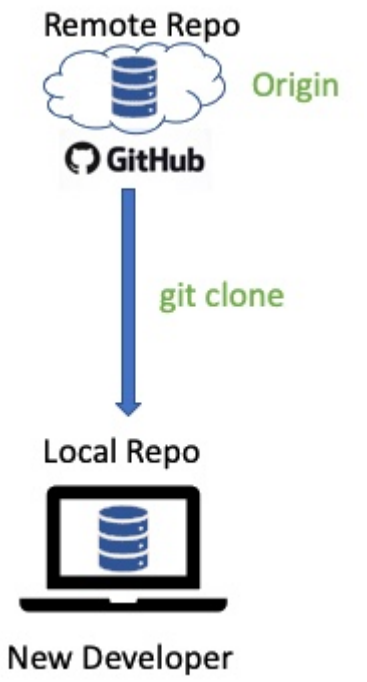
Dağıtılmış Depolar

GitHub gibi dağıtılmış kaynak sürüm kontrol sistemleri, projelerin kod tabanında paralel olarak işbirliği yapan çok sayıda geliştiriciye sahip olmasına izin verir. Bir proje, GitHub.com'da genel veya özel bir depo olarak bulunabilir. Proje üzerinde çalışan her geliştirici, bilgisayarlarında deponun kendi kopyalarına sahip olabilir. Deponun bir geliştiricinin bilgisayarındaki bir kopyası, o geliştirici için yereldir ve dolayısıyla bu geliştirici, bu depoya kendi `local repo`. Deponun GitHub.com'daki kopyası uzak bir sunucudadır ve bu nedenle her geliştirici için bir `remote repo`.

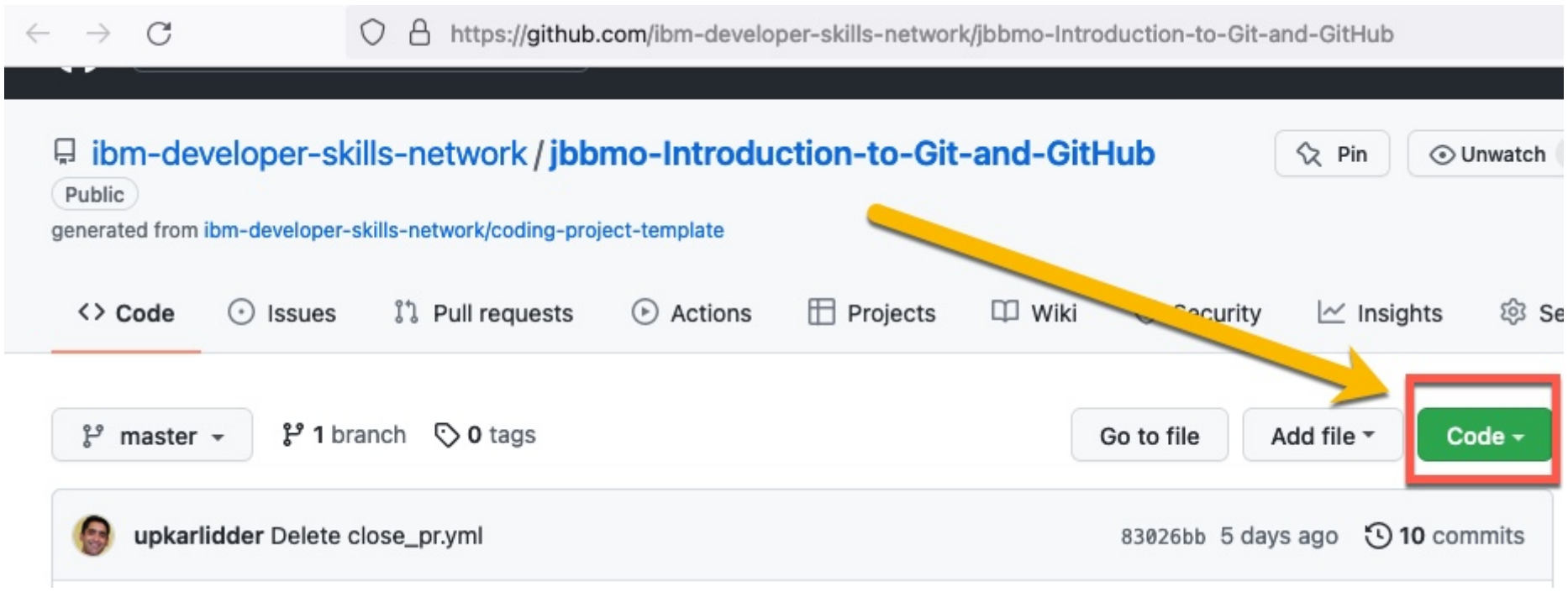


Klon

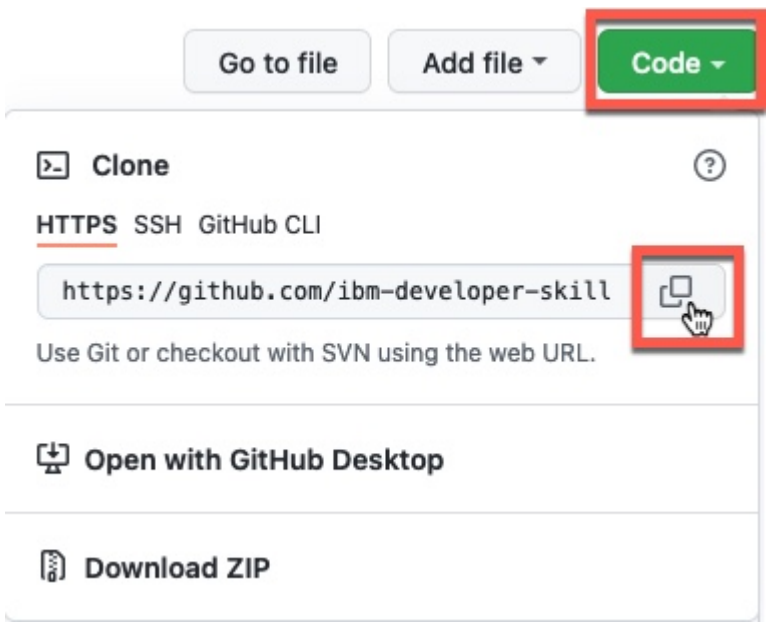
Diyelim ki şimdi yeni bir geliştirici proje üzerinde işbirliği yapmak için ekibe katıldı. `remote repo`Bu geliştirici , işlemin aynı kopyasını oluşturabilir `git clone`. Projenin orijinal olarak klonlandığı uzak depo, aynı zamanda `origin`.



GitHub'daki herhangi bir repo, repoya gidip `Code`düğmesine tıklayarak klonlanabilir.



`git clone URL`Ardından, HTTPS URL'sini kopyalama yeteneği ve ardından yerel makinenizden komut gerçekleştirmek için kopyalanan URL'yi belirtme dahil olmak üzere, uzak deponun tüm kod tabanını çeşitli şekillerde alma seçeneğiniz olacaktır .



Şube Oluşturma ve Değişiklikleri Senkronize Etme

Depoyu yerel makineye klonladıktan sonra, bir geliştirici kod tabanında değişiklikler yapmaya başlayabilir. Bu, özellikler ve geliştirmeler eklemek veya hataları düzeltmek gibi görevler için olabilir. Tipik olarak geliştirici, `git branch`bir dal oluşturmak için kullanır, örneğin `feature1-branch`, bu dalı kullanarak etkin `git checkout`hale getirir ve bu dal içinde - örneğin dosya ekleyerek veya düzenleyerek - değişiklikler yapar. Geliştirici, değişikliklerini genellikle şube içinde, `git add`ardından kullanarak kaydeder `git commit`.

Belirli bir şube için değişiklikler tamamlandıktan sonra, doğrudan ana şubeyle birleştirmek yerine, diğer geliştiricilerin/gözden geçirenlerin şubeyi ana şubeyle birleştirmeden önce değişiklikleri test edebileceği/inceleyebileceği `push`değişikliklerle yeni şubeye gitmek genellikle iyi bir uygulamadır.`origin`

> NOT: Bu senaryoda, `feature1-branch`projede yeni bir geliştirici tarafından geliştirildiğinden, bu geliştiricinin kendi şubesini ana kaynakta ana ile birleştirme erişimi olmayabilir. Aslında, birçok projede, yalnızca proje sorumlularının veya yöneticilerinin ana şubeyle birleşmesine izin verilir veya bazılarında akran değerlendirmesi gerekebilir. Değişikliklerinizin gözden geçirilip ana şube ile birleştirilmesini talep etmek için birçok projede `Pull Request`(PR) sunulması gerekmektedir. Ancak bazı durumlarda, örneğin projede yalnız bir geliştiriciyseniz, bu PR adımı atlanabilir ve kaynak deposuna yazma erişiminiz varsa değişikliklerinizi doğrudan birleştirip iletebilirsiniz.

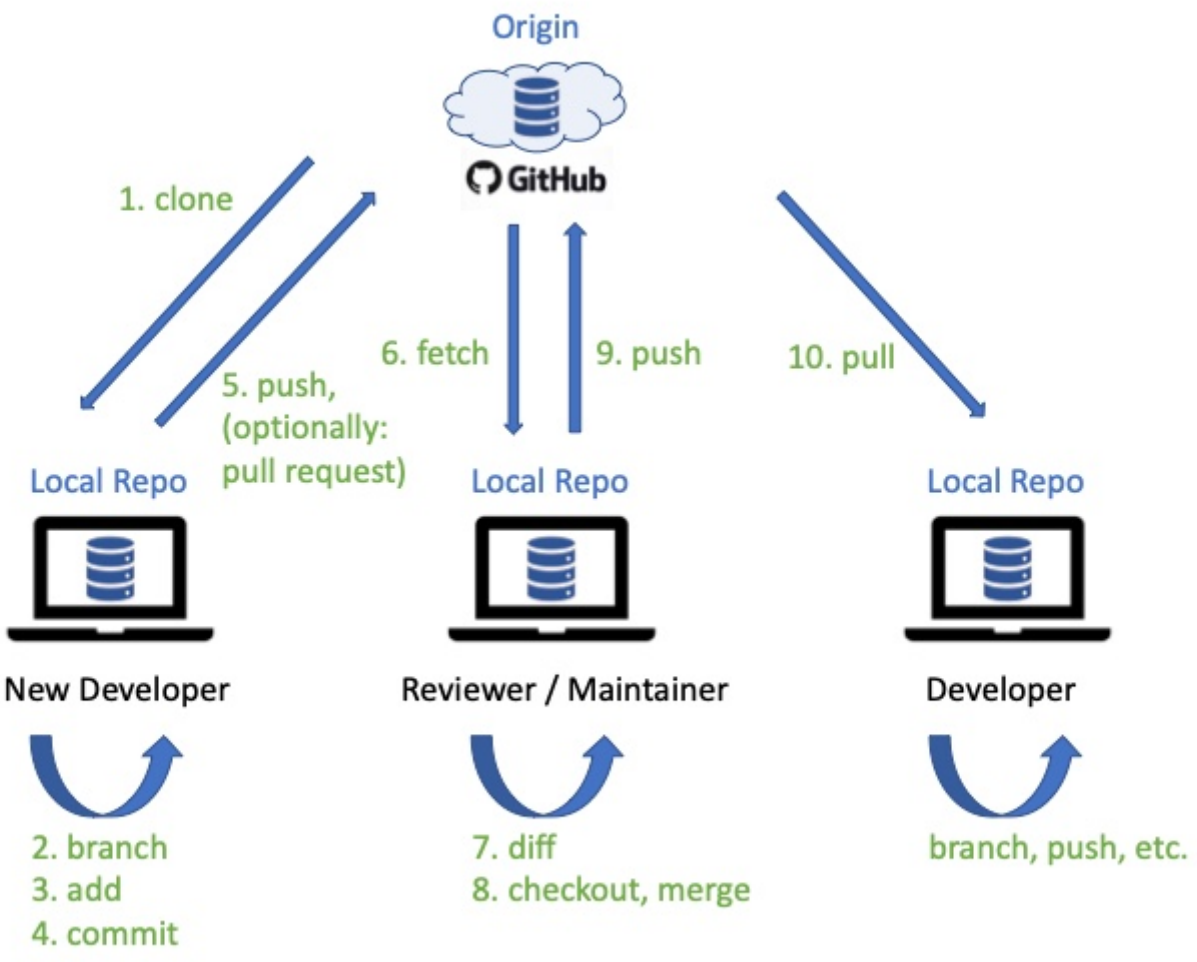
Arada bir, bir geliştirici, `origin`değişiklik yapmak veya başkaları tarafından yapılan değişiklikleri incelemek için temel olarak hizmet vermek üzere deponun en son kopyasını almak isteyebilir. Örneğin, içindeki değişiklikler `feature1-branch`şuraya aktarıldıktan `origin`ve akran geliştirici kodu incelemek istediğinde durum bu olabilir. Komut bu `git fetch`amaç için kullanılabilir.

Komut , `git diff`başkalarının değişiklikleri tanımlaması ve karşılaştırması için kodunuzu gözden geçirmesine yardımcı olabilir. Bir meslektaş gözden geçiren veya proje yürütücüsü değişiklikleri inceledikten ve tatmin olduktan sonra, gözden geçiren kişi `git checkout`ana dalı ve ardından silinebilecek olan `git merge`yeni . `feature1-branch`Dal yerel olarak birleştirildikten sonra, gözden geçiren `git push`kişi güncellenen ana dalı `origin`.

> NOT: `git-remote -v`Komut, push ve fetch değişikliklerini hangi uzak depolarla senkronize ettiğinizi kontrol etmek için kullanılabilir.

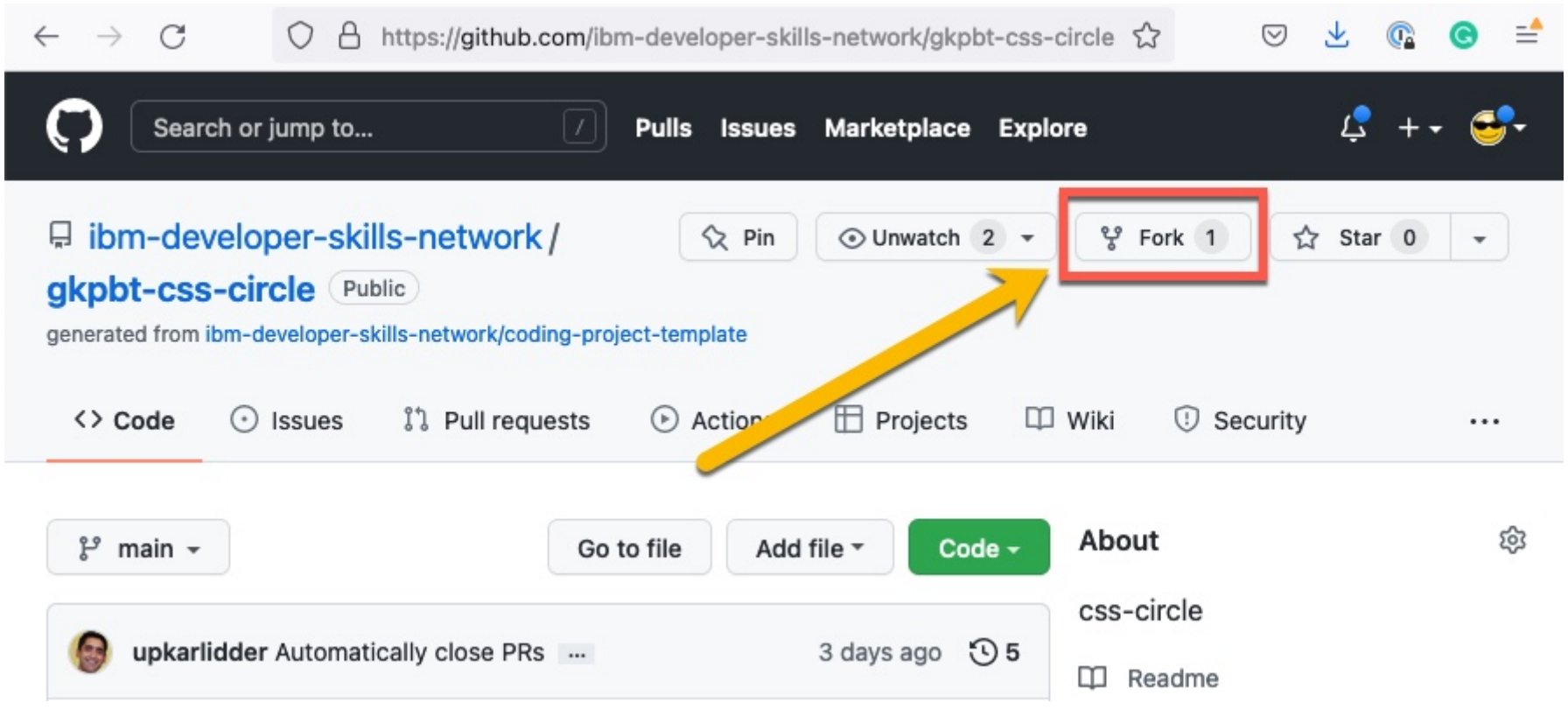
Deponun en son kopyasını almak için başka bir seçenek de `git pull`komutu kullanmaktır. Yürürlükteki komut ve `pull`'nin birleşimidir . Yani, bu tek komutu kullanarak değişiklikleri yerel deponuza getirebilir ve birleştirebilirsiniz. Örneğin, Origin'de ana dalla birleştirilmiş değişikliklerle birlikte güncellenen kod tabanını kullanmak isteyen başka bir geliştirici, yeni bir özellik üzerinde geliştirmeye başlamadan önce , Origin'den güncellenen kod tabanına ve kendi yerel kod tabanına komutu kullanabilir.`fetchmergefeature1git pullfetchmerge`

`clone->branch->merge`Burada açıklanan bu iş akışı aşağıdaki şemada özetlenebilir.



Çatal

`fork`Bir geliştirici, başlangıç noktası olarak başka bir projeyle bir türev proje oluşturmak veya ayrı veya bağımsız bir klon kullanarak bir proje üzerinde çalışmak isterse, geliştirici bir projeyi seçebilir . `Fork`GitHub proje sayfasına gidip sayfanın üst kısmındaki düğmeyi tıklayarak herhangi bir genel projeyi forklayabilirsiniz .

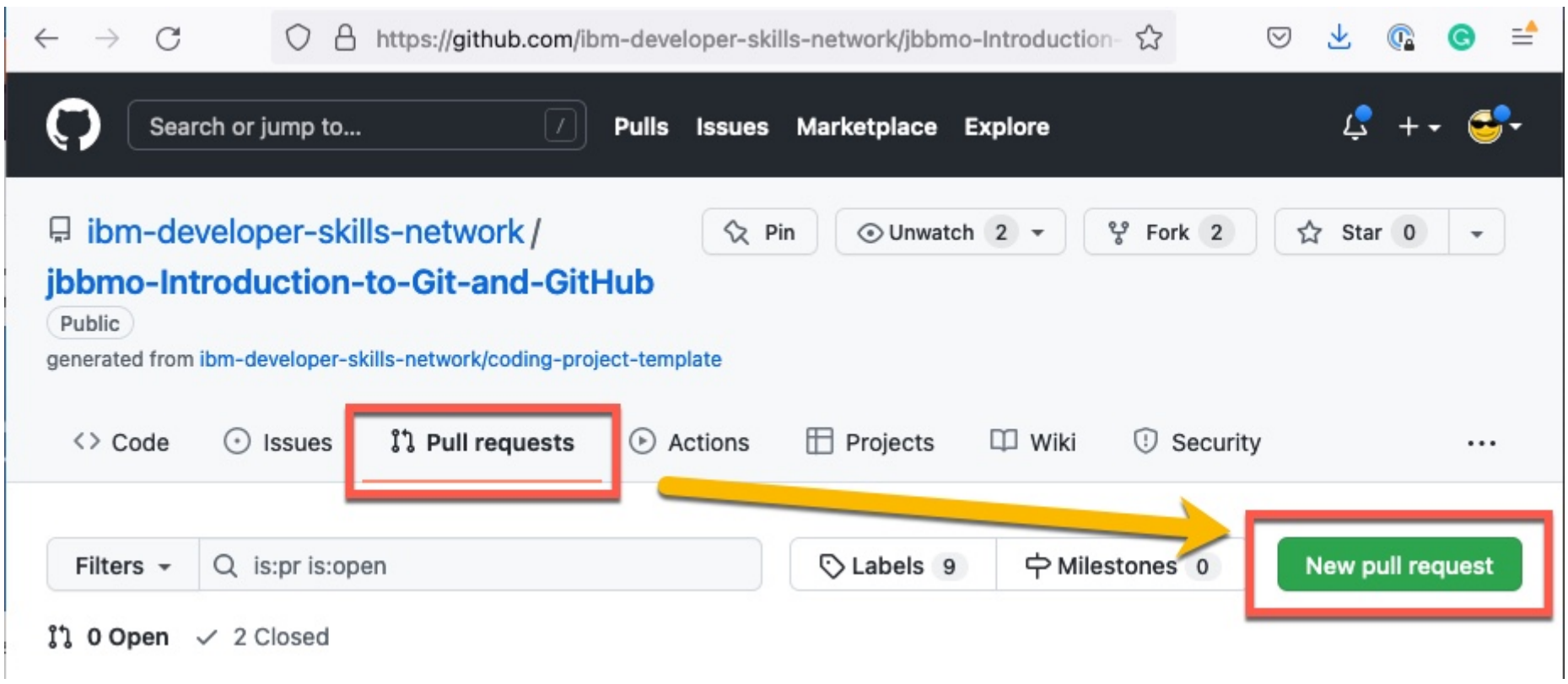


NOT: Çatal seçeneği yalnızca web arayüzü kullanılarak kullanılabilir ve çatal oluşturmak için git komutu yoktur. Bununla birlikte `git clone`, ilgilenirseniz bu okumanın altında belirtilen bir geçici çözüm kullanabilirsiniz.

Çatalı oluşturduğunuz projeye proje denir `upstream`.

Bir proje çatallandıktan sonra, çatala erişimi olan geliştiriciler, daha önce açıklandığı gibi aynı iş akışını kullanarak çatalı güncellemek ve üzerinde değişiklik yapmak için çalışabilirler, yani projenin çatallanmış kopyası artık olur `origin`ve erişimi olan geliştiriciler `origin`şu klonları oluşturabilir: dalları oluşturabilecekleri ve birleştirebilecekleri ve `origin`çekme ve itme kullanarak değişiklikleri senkronize edebilecekleri yerel makinelerinde .

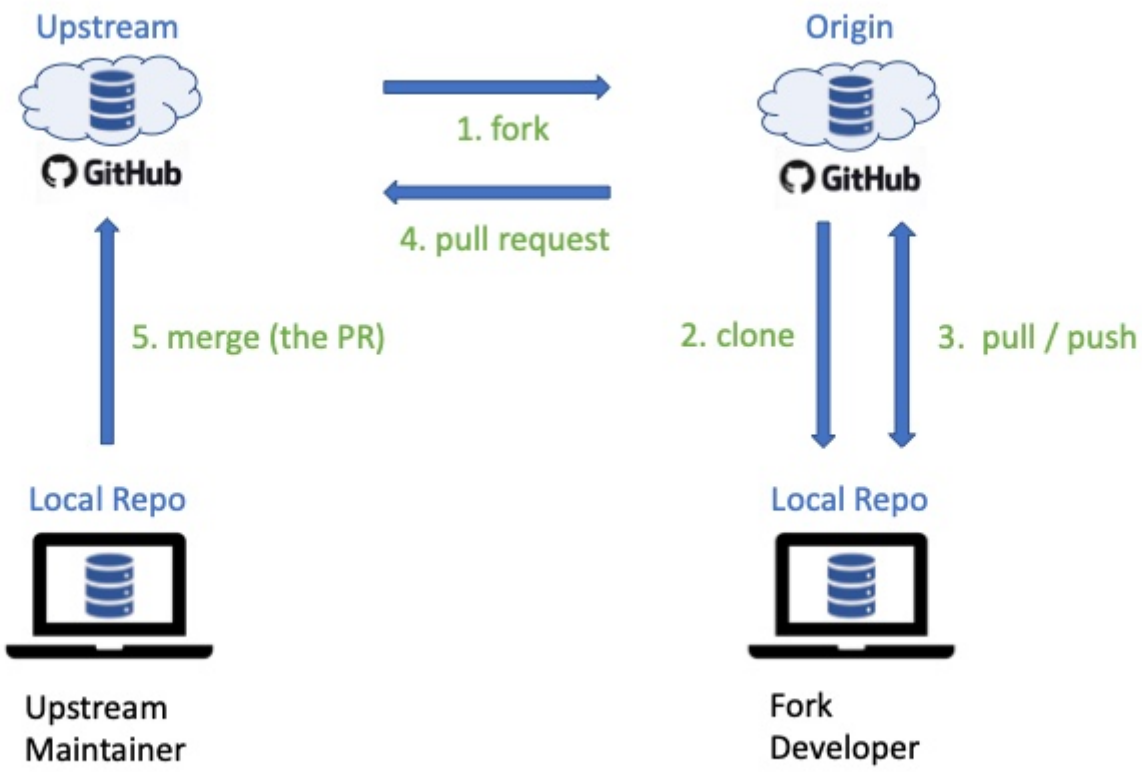
Bununla birlikte, birleştirme ve itme kullanılarak yapılan değişikliklerin senkronizasyonunun, yalnızca geliştiricilerin yazma erişimine sahip olduğu, yani bu durumda proje çatallarına, yani `origin`yerel klonlarını oluşturdukları depolara yapılabileceğini not etmek önemlidir. `upstream`Ancak bir geliştirici , yazma erişimine sahip olmadığı projeye değişiklikleriyle katkıda bulunmak isterse ne olur ? `pull request`Bu durumda , önerilen değişikliklerle birlikte bir veya PR sunabilirler . A , projenin ana sayfasına gidilerek, sekmeye gidilerek ve ardından üzerine tıklanarak `pull request`açılabilir .`Pull RequestsNew Pull Request`



Not: Terim `Pull Request`, `git pull`kullandığınız komutla `fetch`ve `merge`yerel deponuzdaki en son kod tabanıyla karıştırılmamalıdır. A `Pull Request`, adından da anlaşılacağı gibi, yalnızca inceleme talebi ve `pull`önerilen değişikliklerinizdir. Halkla İlişkilerin bir parçası olarak, önerilen değişikliklerin ve uygulamanızın ayrıntılarını sağlarsınız.

Projenin sahipleri `upstream`, PR'daki değişiklikleri gözden geçirebilir ve bunları birleştirip birleştirmemeye karar verebilir. Bazı durumlarda geri bildirim sağlayabilirler (PR'de yorum yaparak) veya PR'ı gönderenden, değişikliklerini en son kod tabanına uygulayarak ve PR'yi yeniden göndererek bazı uyuşmazlık çözümlerini gerçekleştirmesini isteyebilirler.

Burada açıklanan bu `fork->clone->pr`iş akışı aşağıdaki şekilde özetlenmiştir.



Ne zaman çatallanmalı veya klonlanmalı?

Şimdiye kadar çatal ve klon arasındaki farkı biliyor olmalısınız. Öyleyse ne zaman klonlamanız ve çatallanmanız gerektiğini özetleyelim. Tipik olarak, örneğin işbirliği içinde bir kod tabanı geliştiren bir ekibin parçası olarak bir proje deposuna erişiminiz varsa, depoyu klonlayabilir ve çekme ve itme kullanarak deponun yerel kopyanızdaki değişiklikleri senkronize edebilirsiniz.

Bununla birlikte, katkıda bulunmak istediğiniz ancak yazma erişiminiz olmayan veya kendi projeniz için bir başlangıç noktası olarak bir genel proje kullanıyorsanız, projeyi çatallayabilirsiniz. Ardından, çatallı kod tabanını makinenize klonlayarak ve proje çatalınızla çekme-itme senkronizasyonunu kullanarak çatal üzerinde çalışan geliştirme ekibinizle işbirliği yaparak çalışın. Ancak, değişikliklerinizi yukarı akış projesine (içinden çatal aldığınız orijinal proje) geri katkıda bulunmak istiyorsanız, değişikliklerinizi bir çekme isteği kullanarak gönderebilirsiniz.

Özet

Bu okumada şunu öğrendiniz:

- Bir klon, esasen üzerinde değişiklik yapabileceğiniz bir projenin kopyasıdır.
- `git clone`Komutu kullanarak bir projenin yerel kopyasını oluşturabilirsiniz .
- Klonladığınız projeye `origin`.
- `pull`Güncellemeleri `origin`ve `push`değişikliklerinizi ona geri alabilirsiniz .
- Çatal, bir projenin orijinal projeden bağımsız olarak değişiklik yapabileceğiniz ayrı bir kopyasıdır.
- Çatalladığınız projeye proje `upstream`denir
- `Pull Request`Bir (PR) göndererek yukarı akış projesine geri değişiklik önerebilirsiniz.

> Bilginize: Başka bir projenin kod tabanıyla başlamak için olağan iş akışı, önce onu çatallamak ve sonra çatalı klonlamak olsa da, komutu `upstream`kullanarak yerel makinenizden bunu yapmak oldukça uygun olduğu için, projeyi basitçe klonlamak isteyebilirsiniz. `git clone`Bunu yaparsanız, klonladığınız projenin varsayılan olarak `origin`depo olacağını not edeceksiniz. Ancak, klonladığınız yukarı akış deposuna büyük olasılıkla yazma erişiminiz olmadığından, değişikliklerinizi ona aktaramayacaksınız. Merak etme. Komutu kullanarak kaynağı yukarı akış olarak kolayca yeniden adlandırabilir `git remote rename origin upstream`ve ardından `git remote add origin <url>`oluşturduğunuz veya erişiminiz olan yeni bir GitHub deposunun URL'sini işaret etmek için kullanarak yeni bir kaynak ekleyebilir ve bu depoyu çatalın kodunda değişiklik yapmak için kullanabilirsiniz.

Yazar(lar)

Kabalist Ahuja

Diğer Katkıda Bulunanlar

Richard Ye

Değişiklik günlüğü

Tarih	Sürüm	Tarafından değiştirildi	Açıklamayı Değiştir
2022-01-19	1.0	Kabalist Ahuja	İlk sürüm oluşturuldu
2022-01-27	1.1	Richard Ye	Sabit Yazım Hataları